VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Software pro efektivní správu softwarových projektů

Software project manager

2012 Adam Ondrejka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Adam Ondrejka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Systém pro efektivní správu softwarových projektů
Software Project Manager

Zásady pro vypracování:

Cílem práce je navrhnout a implementovat systém pro podporu týmového vývoje softwaru. Výsledný systém by měl mít podobu webové služby s intuitivním rozhraním.

1. Porovnejte dostupné obdobné nástroje.
2. Navrhněte funkcionalitu (např. správa projektů, uživatelů, úkolů a jejich termínů, dokumentace, výměna zpráv, bug-tracking, správa zdrojových kódů pomocí GIT nebo SVN).
3. Navržený systém implementujte (Python, Django, MongoDB, HTML5, Ajax).
4. Funkčnost výsledného softwaru ověřte na modelové ukázce.

Seznam doporučené odborné literatury:

[1] Anders, M. Python 3 Web Development Beginner's Guide. 2011. ISBN 9781849513746.
[2] Alchin, M. Pro Python. 2010. ISBN 9781430227571.
[3] Wysocki, R. K. Rapid Development: Taming Wild Software Schedules. 1996. ISBN 9781556159008.
[4] Stellman, A., Greene, J. Applied Software Project Management. 2005. ISBN 9780596009489.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Tomáš Fabián**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 2012 .....

Na tomto místě bych rád poděkoval vedoucímu bakalářské práce za cenné rady a nápady.

Abstrakt

Tato bakalářská práce se zabývá analýzou a implementací webové aplikace pro správu softwarových projektů. Obecně pojednává o projektovém řízení, porovnává některé dostupné webové služby pro správu projektů. V analýze jsou přesněji specifikovány požadavky na finální systém, podle kterých je provedena datová a funkční analýza. Popisuje samotný návrh a implementaci webové aplikace, použité technologie, strukturu aplikace, ukázky z realizace a popis produkčního nasazení.

Klíčová slova: správa projektů, Python, Django, MongoDB, HTML5, CSS3, webová aplikace, nerelační databáze

Abstract

This thesis deals with the analysis and implementation of a web application for managing software projects. Generally deals with project management, compares some of the available web services for managing projects. The analysis accurately specifies requirements for a final system, which they are made and functional data analysis. Describes design and implementation of web applications, technology, application architecture, examples of implementation and a description of the production deployment.

Keywords: project management, Python, Django, MongoDB, HTML5, CSS3, web application, non-realation database

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XM
BSON	–	Binary JavaScript Object Notation
CSS	–	Cascading Style Sheets
DFD	–	Data Flow Diagram
DOM	–	Document Object Model
FTP	–	File Transfer Protocol
HTML	–	Hyper Text Markup Language
HTTP	–	Hyper Text Transfer Protocol
IAAS	–	Infrastructure as a Service
IT	–	Informační Technologie
JSON	–	JavaScript Object Notation
MB	–	Mega byte
MS	–	Microsoft
MVC	–	Model View Controller
NoSQL	–	no Structured Query Language - označení pro nerelační databáze
ORM	–	Objektově relační mapování
OS	–	Operační systém
SAAS	–	Software as a Service
SQL	–	Structured Query Language
SSH	–	Secure Shell
URL	–	Uniform Resource Locator
VPS	–	Virtualní privátní server
XML	–	Extensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce se zabývá správou softwarových projektů. Ať už na vývoji pracuje jeden člověk nebo celý tým odborníků, je vhodné svůj projekt pro zvýšení produktivity efektivně spravovat.

Mým cílem je vytvořit nástroj pro rozdělování úkolů, sledování termínů vázajících se k projektu, nahlašování chyb, výměnu zpráv, uložiště souborů, úschovu dokumentů a správu zdrojových kódů. Určen bude především pro jednotlivce a menší vývojové týmy, které nemají prostředky pro správu vlastního serverového řešení, popřípadě lidem, kterým tento způsob projektového managementu bude vyhovovat. Vycházím především z vlastních poznatků a zkušeností, které jsem získal při práci na několika projektech. Stanoviska jsou založeny na vlastním bádání a analýzách daných problémů, ale také ze zkoumání již dostupných nástrojů a následné inspirace. I přesto, že trh je již podobnými správci projektů poměrně zasycen, dá se vytvořit produkt odlišný od ostatních a s cílením na specifickou skupinu uživatelů.

Následující kapitola především stanovuje a popisuje cíle této bakalářské práce. V další části práce se dočtete, co vlastně znamená projekt, jeho řízení a jak se plánuje. Dozvíte se něco o roli projektového manažera, rozdělování úkolů, Ganttově diagramu a způsobu sledování chyb u softwarových produktů. A nebude chybět ani porovnání vybraných webových aplikací pro správu projektů.

Čtvrtá kapitola se věnuje analýze aplikace. Dočtete se zde o požadavcích na výsledný systém, kdo s ním bude pracovat, proč jej vůbec vytvářet, jaké požaduje vstupy a jaké můžete očekávat výstupy této webové služby. A také nahlédnete pod povrch datové a funkční analýzy a na návrh uživatelského rozhraní.

Pátá kapitola již popisuje samotný návrh a implementaci konečného systému. Dozvíte se zde více informací a použitých technologiích, jako jsou nerelační databáze MongoDB či webový framework Django a také, že je vytvořena pomocí dynamického jazyku Python. Zjistíte něco o produkčním nasazení aplikace, řešení hostingu zdrojových souborů pomocí verzovacího systému Git nebo o budoucích možnostech škálovatelnosti celého systému. V poslední řadě se můžete těšit na modelovou úkazku malého vývojového týmu, který si pro zvýšení efektivity práce na jejich projektech vybral právě tuto bakalářskou práci.

2 Stanovení a popis cílů

Cílem práce je navrhnout a implementovat systém pro podporu týmového vývoje softwaru. Výsledný systém by měl mít podobu webové služby s intuitivním rozhraním.

1. Porovnejte dostupné obdobné nástroje.
2. Navrhněte funkcionalitu (např. správa projektů, uživatelů, úkolů a jejich termínů, dokumentace, výměna zpráv, bug-tracking, správa zdrojových kódů pomocí GIT nebo SVN).
3. Navržený systém implementujte (Python, Django, MongoDB, HTML5, Ajax).
4. Funkčnost výsledného softwaru ověřte na modelové ukázce.

Prvním cílem je analyzovat a porovnat dostupné konkurenční nástroje pro správu projektů. Při porovnání bude klíčové rozdělování úkolů, správa uživatelů, sledování chyb v projektu, verzování zdrojových kódů, správa souborů a případné způsoby komunikace mezi uživateli. Srovnávat budu jen podobné webové aplikace.

Dále analyzuji požadavky na požadovaný systém. Obsahem budou funkční i nefunkční požadavky, proč vůbec tuto aplikaci vytvářet, popíši, kdo se systémem bude pracovat a definuji také vstupy a výstupy aplikace. Samotná analýza se pak skládá z části datové a funkční. Datová analýza obsahuje lineární zápis, konceptuální model a třídní diagramy. Ve funkční analýze je obsažen kontextový diagram, diagram užití a diagramy toku dat. Následuje možný návrh uživatelského prostředí systému a volba konečného jména aplikace.

Třetím cílem je implementace navrženého systému. Uživatelská část bude vytvořena pomocí HTML5 a skriptovacího jazyku Javascript pro interaktivnější komunikaci s uživatelem. Jádro - serverová část - aplikace bude realizováno pomocí programovacího jazyku Python a webového frameworku Django. Jako databázový systém bude zvolen nerelační MongoDB, verzování a správu zdrojových kódů zajistí distribuovaný systém Git. Rovněž se zaměřím na realizaci některých užitečných funkcí, jako je např. Ganttův diagram nebo řešení zabezpečeného přístupu ke git repozitářům. Celé řešení poběží na linuxovém serveru s pomocí serverů Apache a nginx.

A nakonec otestuji funkčnost aplikace na modelové ukázce. Malá firma zabývající se vývojem webových prezentací a jednodušších aplikací potřebuje zefektivnit práci na svých projektech. Tým skládající se ze 4 lidí bude veden projektovým manažerem a obchodníkem v jedné osobě. Práci rozděluje na základě požadavků od klienta mezi další 3 členy týmu - grafika, kodéra šablon a programátora. Dosavadní řešení komunikace a rozdělování úkolů pomocí emailů a sdílení dat oddělenými hostingovými službami firmě nevyhovuje, proto se rozhodnou vyzkoušet toto řešení.

3 Správa a řízení projektů

Softwarové projekty mohou vyvíjet jednotlivci, tým či dokonce více týmů o několika lidech, kteří se nemusí ani osobně potkat. Zdárné dokončení v takovém případě není jen doménou samotných vývojářů, ale také jejich správným vedením a koordinací. Tímto je zpravidla pověřen takzvaný „projektový manažer", který bývá pověřen největším množstvím pravomocí a zároveň na něm leží veškerá zodpovědnost za úspěšné dokončení projektu.

3.1 Co je vlastně projekt?

Již několikrát jsem zmínil slovo „projekt", ale co vlastně znamená? Dá se najít mnoho vyložení významu, osobně se mi nejvíce zamlouvá definice z knihy PMBOK [1], která zní: „projekt je dočasné úsilí s cílem vytvořit unikátní produkt nebo službu". Projekt tedy musí být časově omezen a to jak začátkem tak koncem, jeho dokončením musí vzniknout jasný výsledek a je také omezen zdroji, které jsou pro realizaci k dispozici. Při práci na projektu samozřejmě vznikají náklady a také rizika, které je vhodné předem odhadnout a posoudit.

3.2 Plánování projektu

Před samotným plánováním práce na projektu tedy potřebujeme znát cíl a termín zahájení nebo ukončení. Cíl může být konečný produkt nebo stav, kterého chceme my nebo zákazník dosáhnout (např. zvýšení návštěvnosti e-shopu o 60%).

V dalším kroku se konečný cíl rozdělí na cíle menší, rozfázují se a stanoví se milníky. Jedná se o body vyznačující kritické místo v plánu projektu, doba trvání milníku je nulová. Jde především o informativní body vyjadřující, zda průběr realizace projektu odpovídá plánování. Jednotlivé fáze se poté rozdělí na konkrétní úkoly a ty se přidělí ke zdrojům - lidem.

Projektový manažer by měl mít především reálná očekávání a odhadnout náklady na realizaci projektu. Pokud výrazně převyšují dostupné prostředky, popřípadě očekávání jsou jasně nereálná, nemá smysl se do realizace projektu vůbec pouštět. Stejně tak by měl vytvořit prvotní dokument obsahující specifikaci požadavků a již teď objevené problémy a rizika, aby s nimi zadavatel v budoucnu počítal.

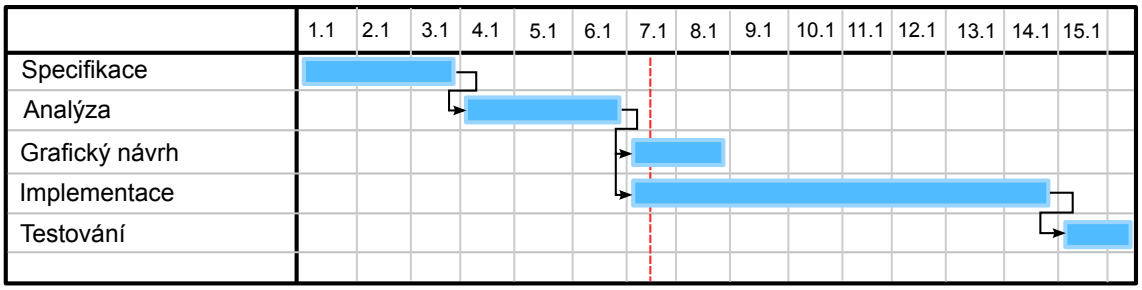
3.3 Rozdělování úkolů

Jak jsem uvedl výše, úkol se dá považovat za nejmenší část cíle. Aby byl takový úkol splněn, musí mít uvedeny určité vlastnosti. Mezi první z nich patří „co" se má údělat, respektive čeho se má dosáhnout, aby úkol bylo možno označit za dokončený. Dále je třeba rozhodnout, „kdo" jej má udělat. Toto záleží na schopnostech člověka úkol vyřešit (grafik pravděpodobně nezvládne naprogramovat informační systém) a také na jeho časové vytíženosti. Určitě musíme také určit, „kdy" má být dokončen. Termín nemusí být fixní, může být závislý na jiném úkolu. S časem souvisí také další vlastnost a sice začátek práce

na úloze. Nejedná se o povinný atribut, je vhodné jej ale uvádět pro orientaci celkové práce na úkolu. Při projektovém plánování je dále vhodné uvézt „prioritu“ a průběžně uvádět „stav“ k dokončení. Pro správné pochopení úlohy je samozřejmostí srozumitelně sepsaná „dokumentace“.

3.4 Ganttův diagram

Pro lepší přehlednost plánování úkolů je vhodné použít vizualizační techniky. Mezi velmi používanou a oblibenou metodu patří Ganttův diagram. Jde o typ pruhového diagramu pojmenovaný po jeho průkopníkovi během první světové války H. L. Ganttovi. Používá se pro plánování posloupnosti činností v čase. Horizontální osa diagramu je rozdělena do stejně dlouhých časových jednotek, zpravidla dnů. Vertikální osa označuje jednotlivé činnosti, pro jednu vždy jeden řádek. Samotná plocha diagramu je pak tvořena různě velkými obdelníky, které označují dobu trvání činnosti. Levé ohraničení znázorňuje začátek a pravé ohraničení konec předpokládané doby. Takto znázorněný diagram přehledně ukazuje návaznost jednotlivých činností, v našem případě hlavně úkolů. Moderní pojetí Ganttova diagramu může také znázorňovat návaznost mezi úkoly vykreslováním lomených šipek. Používá se také míra dokončení činnosti v procentech nebo svislé linky pro znázornění aktuálního data či vymezení různých milníků. Ukázka je na obrázku 1.



Obrázek 1: Ganttův diagram

3.5 Plánování softwarových projektů

Proces plánování softwarových projektů se od projektů z jiných oblastí v podstatě neliší. Výsledkem je rovněž produkt, který má spatřit světlo světa, jen má zpravidla formu nějaké aplikace. Zásadní rozlišení můžeme nalézt v nástrojích pro jejich řízení. Nástroje zaměřené speciálně pro vývoj softwaru obsahují přídané funkce usnadňující samotnou realizaci projektu. Může se jednat například o možnost sledování chyb softwaru nebo správu zdrojových kódů produktu.

3.5.1 Sledování chyb systému

Systém sledování chyb nebo-li anglicky „bug tracking system“ bývá zpravidla softwarová aplikace, navržena pro sledování nahlášených chyb v softwarovém produktu. Typicky je

integrován právě do aplikací pro správu projektů. Princip takového systému je následující. Tester aplikace popřípadě její uživatel nahlásí - zadá objevenou chybu do systému pro sledování chyb. Součástí zadání by měl být co nejdetailnější popis chyby a verze systému, kde byla chyba objevena a její závažnost. Tedy, zda se jedná o kritickou chybu nebo jen například o nějaký kosmetický nedodělek v grafickém návrhu. Takto zadaný problém se pak v systému objeví zodpovědné osobě, která chybu zadá další osobě k opravě, popřípadě ji ověří nebo si vyžádá další detailnější informace. Vývojář po opravě změní status chyby podle jím dosaženého výsledku. Systém na základě historie opravy chyb může třeba generovat reporty o produktivitě jednotlivých vývojářů.

3.5.2 Správa a verzování zdrojových kódů

Pro správu zdrojových kódů lze použít různé metody. Pokud má vývoj softwaru na starost jeden člověk, pravděpodobně si vystačí s vlastním pevným diskem nebo zálohou na přenosném médiu, popřípadě na vzdáleném serveru, a verzování by mohl řešit přejmenováním kopií. Pří vývojové práci je toto řešení ovšem z mnoha důvodů nevhodné. Proto nastupují na scénu tzv. systémy pro verzování zdrojových kódů. Ty umožňují kolaborativní vývoj a především snadné vytváření různých verzí kódů. Dnes se užívají dva typy, centralizované a distribuované. U centralizovaných systémů jsou data ukládány na jediný server a pro provedení veškerých změn a zkoumání verzí je nutno být s ním ve spojení, což poněkud zpomaluje. Na základě toho vznikly právě systémy distribované, s kterými nemusí vývojář neustále komunikovat. Pouze si stáhne požadovanou verzi na lokální počítač a na server nahrává až provedené změny.

3.6 Nástroje pro řízení projektu

Po naplánování již následuje proces řízení projektu. Pod tím si můžeme představit proces realizace stanovených akcí pro dosažení cíle, lze jej také charakterizovat jako účinné dosahování efektivních změn. Způsob řízení záleží především na rozsahu a typu projektu. V některých případech vystačí pouze lidská paměť či poznámkový blok, u složitějších a náročnějších projektů už bude nutno sáhnout po sofistikovanějších nástrojích a řešeních. Zvolený nástroj by měl umožňovat především správu úkolů a zdrojů.

Software pro řízení projektu se zpravidla dělí na desktopové aplikace a webové služby.

Desktopové aplikace

Klasické desktopové programy, které se instalují na klientské počítače. Podle softwaru potom záleží, jestli umožňují pouze správu projektu na jednom počítači v lokální databázi (např. Planner pro OS Linux) nebo se data ukládají na vzdáleném serveru a je možné k nim přistupovat vzdáleně (např. MS Project). Mezi výhody patří provoz na vlastní infrastruktuře, možnosti uživatelského rozhraní, které nabízí operační systém. Mezi nevýhody potom přístup pouze z počítače s nainstalovaným programem.

Webové služby

Velký „boom“ zažívají webové služby a nevyhly se ani tomuto odvětví. Jako uživatel máte na výběr, zda na vlastním serveru provozovat zakoupené či open source řešení, nebo využít dostupných SAAS a pouze vytvořit účet za většinou paušální poplatek. Výhoda internetového řešení je především v přístupu z kteréhokoliv zařízení připojeného k internetu, všechny data na jednom místě a mnoho služeb navíc poskytuje nativní mobilní aplikace, takže si aktuální stav projektu můžete zkontrolovat téměř kdykoliv a kdekoliv. Jako nevýhodu bych uvedl o něco vyšší pravděpodobnost odcizení dat a nutnost být pro práci připojen k internetu.

3.7 Přehledová studie dostupných webových služeb

Protože výsledkem této bakalářské práce je webová aplikace, budu dále analyzovat pouze konkureční webové aplikace. Výběr kvůli velkému množství neobsahuje všechny služby na trhu, jedná se o osobní výběr.

Basecamp.com

Jedna z nejznámějších služeb řízení projektů. Určen především pro běžné uživatele, vyznačuje se jednoduchým a přehledným prostředím. Umožňuje správu úkolů, time tracking, výměnu zpráv mezi uživateli, správu souborů, diskuzi a komentáře, připomínání událostí. Nezaměřuje se pouze na IT vývojáře, z tohoto důvodu chybí bug-tracking a verzovací systém pro správu zdrojového kódu. K dispozici je několik placených úrovní, zdarma má uživatel k dispozici možnost spravovat sám jeden projekt [3].

Projectial.com

Původem česká služba s velmi pěkným uživatelským rozhraním. Umožňuje spravovat úkoly, stanovovat milníky k projektu, vytvářet společnosti. Rovněž obsahuje výměnu zpráv ve formě diskuze a možnost komentovat úkoly. Aplikace je zatím dostupná zdarma, autor slibuje po dodání všech funkcí zpoplatnění přístupu. Chybí možnost správy souborů, dokumentů, zdrojových kódů a bug-tracking [4].

Zoho Projects

Velmi mocný a schopný nástroj. Pro jeho komplexnost najde uplatnění především u velkých společností a rozsáhlých projektů. Spravuje úkoly dokumenty, umožnuje bug a time tracking, nabízí různé statistiky a grafy, psaní dokumentů, wiki, projektový kalendář, management souborů, synchronizaci s Google Kalendářem. Nebosahuje správu zdrojových kódů. Ceny začínají na $20 měsíčně[1] za základní verzi, za další funkce se celková cena navyšuje. Pokud firma nebo vývojář nemá problém si za kvalitní službu připlatit, tuto webovou aplikaci rozhodně doporučuji [5].

[1] ceník přístupný v březnu 2012

	Basecamp	Projectial	Zoho projects	Github
Správa úkolů	•	•	•	o
Bug tracking	o	o	•	•
Dokumenty	•	o	•	•
Soubory	•	o	•	o
Hostování zdrojových kódů	o	o	o	•
Kalendář	•	o	•	o
Zprávy	•	•	•	•

Tabulka 1: Srovnání konkurenčních webových aplikací

Github.com

Nejedná se přímo o aplikaci pro řízení projektů, ale mohla by svou funkčností některým uživatelům vyhovovat. Primárně se jedná o službu určenou k hostování zdrojových kódů pomoci verzovacího systému Git. Obsahuje ale také bug-tracking, správu uživatelů, možnost dokumentace pomocí wiki. Bohužel chybí rozdělování úkolů [6].

4 Analýza

Analýza požadavku a řešení je důležitá část při vývoji softwaru, popisuje hlavně jeho datovou strukturu a proces při vývoji. První krokem je definice požadavku a očekávání, dále vytvoření konceptuálního modelu, který popisuje návrh datové struktury, atributy a vztahy mezi entitami. Nakonec následují diagramy v jazyce UML o případech užití, třídní diagramy a diagramy datových toků.

4.1 Analýza požadavků

Cílem je vytvořit systém pro správu vývoje softwarového projektu určený především začínajícím vývojářům a menším firmám. Se systémem bude pracovat osoba, která bude mít na starost správu projektu (dále jen „správce projektu“), lidé podílející se na vývoji projektu (dále jen „vývojáři“) a administrátoři služby (dále jen „administrátoři“). Do samotné aplikace budou mít přístup jen registrovaní uživatelé. Z tohoto důvodu je nutné vytvořit systém na registraci, přihlašování a následnou kontrolu práv uživatele. Po registraci nového uživatele má uživatel možnost vytvořit si vlastní projekt nebo stát se součásti týmu pracujícím na již existujícím projektu.

Hlavní myšlenkou tohoto systému je přístup k vývoji založeném na komunikaci a dokumentaci. Cílem je dát vývojářům možnost zachytit myšlenky, nápady a poznatky již od prvního návrhu projektu. Stejně tak umožnit vzájemnou komunikaci, rozdělování úkolů, reportování chyb, vytyčení milníků (termínů) pro vývojáři definovanými fázemi projektu. Dále bude také vývojářům umožněno spravovat zdrojový kód pomocí verzovacího systému GIT.

4.2 Specifikace požadavků

Specifikací požadavků bychom měli vyvodit konkrétní zadání celého systému. Pokládáním správných otázek a následnými odpověďmi dostaneme všechny důležité informace potřebné k následné analýze systému.

4.2.1 Proč tento systém?

K vytvoření tohoto systému mě vedla vlastní potřeba po podobném nástroji. Během vývoje jednoho projektu s přáteli jsme narazili na mnoho problémů při neorganizovaném vývoji. Na základě těchto zkušeností jsem vytvořil model aplikace, která by efektivně pokryla většinu objevených problémů.

4.2.2 Kdo bude se systémem pracovat?

Se systémem budou pracovat lidé podílející se jakýmkoliv způsobem na vývoji projektu, především softwarového. Dále jej budou obsluhovat administrátoři služby. Podmínkou pro využívání služby je registrace uživatele.

4.2.3 Vstupy

- **Uživatel** – po uživatelích budeme vyžadovat emailovou adresu, zvolení vlastního hesla, zadání jména a příjmení
- **Projekt** – registrovaný uživatel bude mít možnost vytvoření projektu. U něj je nejdůležitější jej pojmenovat a v případě vývoje ve více lidech přiřadit další uživatele – vývojáře
- **Úkoly** – správce projektu, případně vývojář, pokud je mu to umožněno, přiřadí k projektu úkoly, které je třeba dokončit. Každý úkol může mít stupeň důležitosti, začátek a termín dokončení
- **Termíny/události** – správce projektu/vývojář může k projektu přidat důležité termíny či události s ním spojené. Na základě těchto údajů se bude tvořit časová osa (timeline)
- **Dokumenty** – uživatel bude moci nahrávat k projektu různé dokumenty
- **Soubory** - uživatel bude moci nahrávat k projektu soubory
- **Nápady** – uživatelé budou moci zaznamenávat nápady a myšlenky k vývoji projektu
- **Komentáře** – vývojáři budou moci komentovat úkoly, události, nápady, pokud správce projektu tuto možnost povolí
- **Zprávy** - vývojáři si budou moci posílat interní zprávy

4.2.4 Výstupy

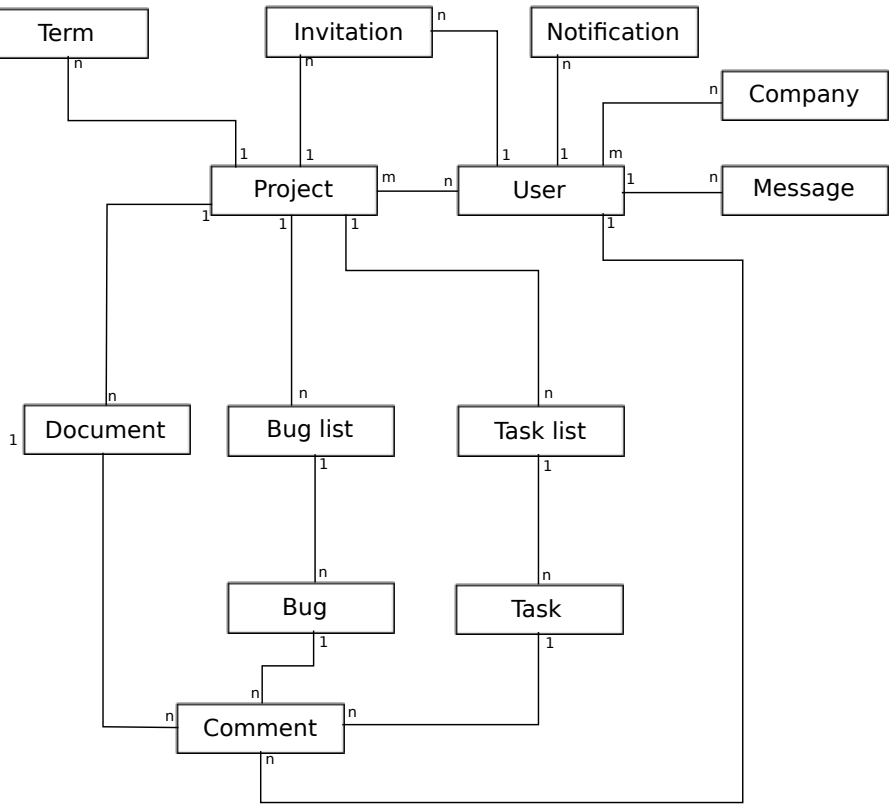
- Uživatel má možnost psát a přijímat zprávy od jiných uživatelů. Dále si může upravit nastavení vlastního profilu.
- Uživateli se zobrazí úkoly, které mu byly přiděleny, důležité termíny a události.
- Uživatel má přístup ke všem mu viditelným dokumentům.
- Správce projektu má přehled o celém projektu.
- Administrátor má obecný přehled o projektech a uživatelích. Má přístup ke statistikám systému. NEMÁ přístup k interním u a citlivému obsahu o projektech a uživatelích!

4.3 Datová analýza

Tato část popisuje datovou analýzu projektu. Bude obsahovat lineární zápis a konceptuální schéma, datový slovník není vzhledem k použité databázi potřeba, proto jej vynechám.

4.3.1 Konceptuální model

Konceptuální model vyjadřuje model reality bez ohledu na budoucí způsob řešení. Vede k nalezení všech entit, vztahů a atributů a popisuje data bez ohledu na použitý databázový systém. Je znázorněn na obrázku 2.



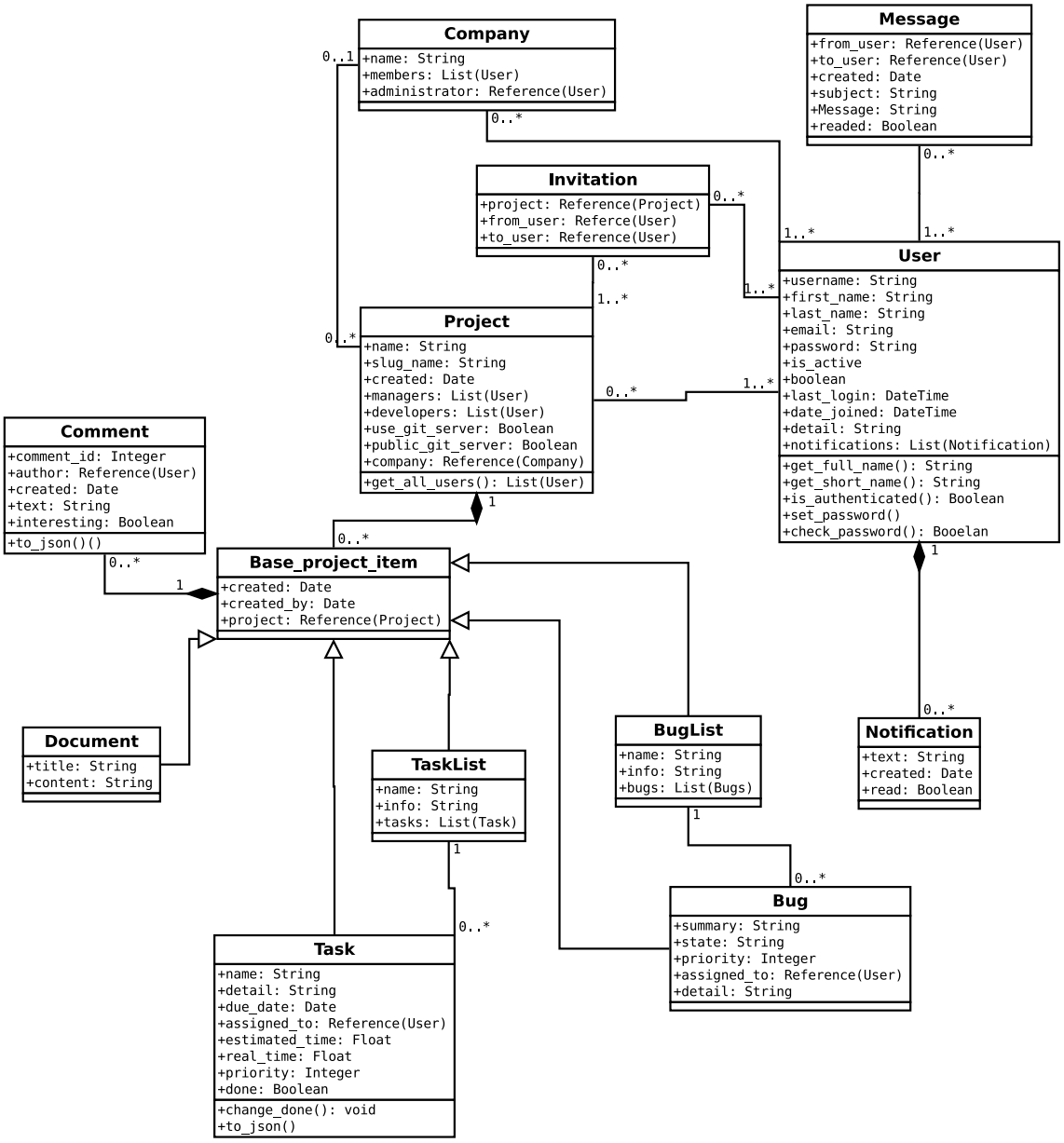
Obrázek 2: Konceptuální datový model

4.3.2 Lineární zápis

Lineární zápis je znázorněn v tabulce 2. Protože je použit nerelační databázový systém MongoDB, znázorněné indexy jsou jen ilustrativní. Ve skutečnosti žádné spojení v rámci databáze nevzniká, pouze se indexují dané atributy. Samotné spojení je nutno provádět až v aplikaci. Navíc je vhodnější mít entity databáze v denormalizované formě, správně normalizovaná je naopak nevhodná.

4.3.3 Třídní diagram

Třídní diagram znázorňuje statické struktury systému prostřednictvím tříd a vztahů mezi nimi. Vztahy mezi třídami objektů jsou znázorněny na obrázku 3.



Obrázek 3: Třídní diagram

User	**id** , username, first_name, last_name, email, password, is_active, is_supersuser, last_login, date_joined, detail, is_public, notifications, last_updated
Project	**id**, name, slug_name, created, *managers*, *developers*, terms, term_counter, is_public, use_git_server, public_git_server, activities
Term	**term_id**, title, detail, due_date, created
Message	**id**, *from_user*, *to_user*, created, subject, message, readed
TaskList	**id**, *project* , *created_by*, created, name, info, *tasks*
Task	**id**, *project* , *created_by*, created, name, detail, due_date, comments, comments_counter, *assigned_to*, estimated_time, priority, done
BugList	**id**, *project* , *created_by*, created, info, *bugs*
Bug	**id**, *project* , *created_by*, created, summary, state, priority, *assigned_to*, comments, comments_counter, detail
Comment	**comment_id**, *author*, created, text, interesting
Doc	**id**, *project* , *created_by*, created, title, content
Invitation	**id**, *project*, *from_user*, *to_user*
Notification	**id**, *user*, created, text, read
Company	**id**, *administrator*, *members*, name

Tabulka 2: Lineární zápis

4.4 Funkční analýza

Cílem funkční analýzy je zamyšlení nad tím, jak specifikované požadavky realizovat pomocí funkcí a jak popsat všechny operace s daty v navržené databázi.

4.4.1 Seznam funkcí

Uživatel / vývojář:

- registruje se
- spravuje svůj profil
- spravuje své publikované příspěvky (úkoly, nápady, dokumenty, chyby...)
- má přístup k verzovacímu systému
- komentuje
- posílá zprávy

Správce projektu navíc:

- vytváří a spravuje projekt
- vymezuje uživatele pracující na projektu

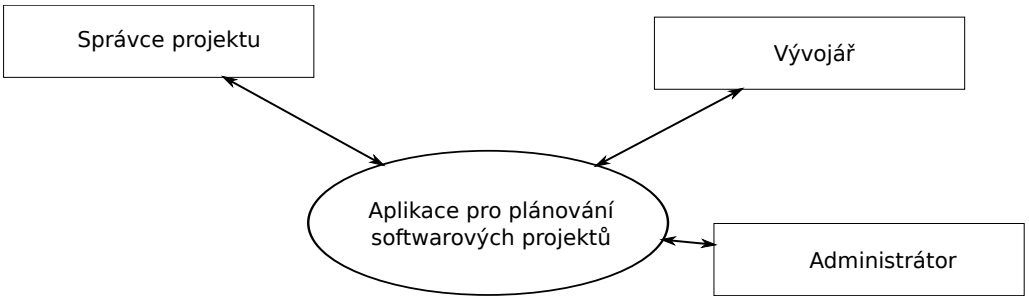
- přiřazuje vývojáře k úkolům
- spravuje dokumenty
- spravuje všechny publikované příspěvky (úkoly, nápady, dokumenty...)

Administrátor

- na vyžádání upravuje profily uživatelů
- přihlašuje se
- odhlašuje se
- sleduje stav a statistiky systému

4.4.2 Kontextový diagram

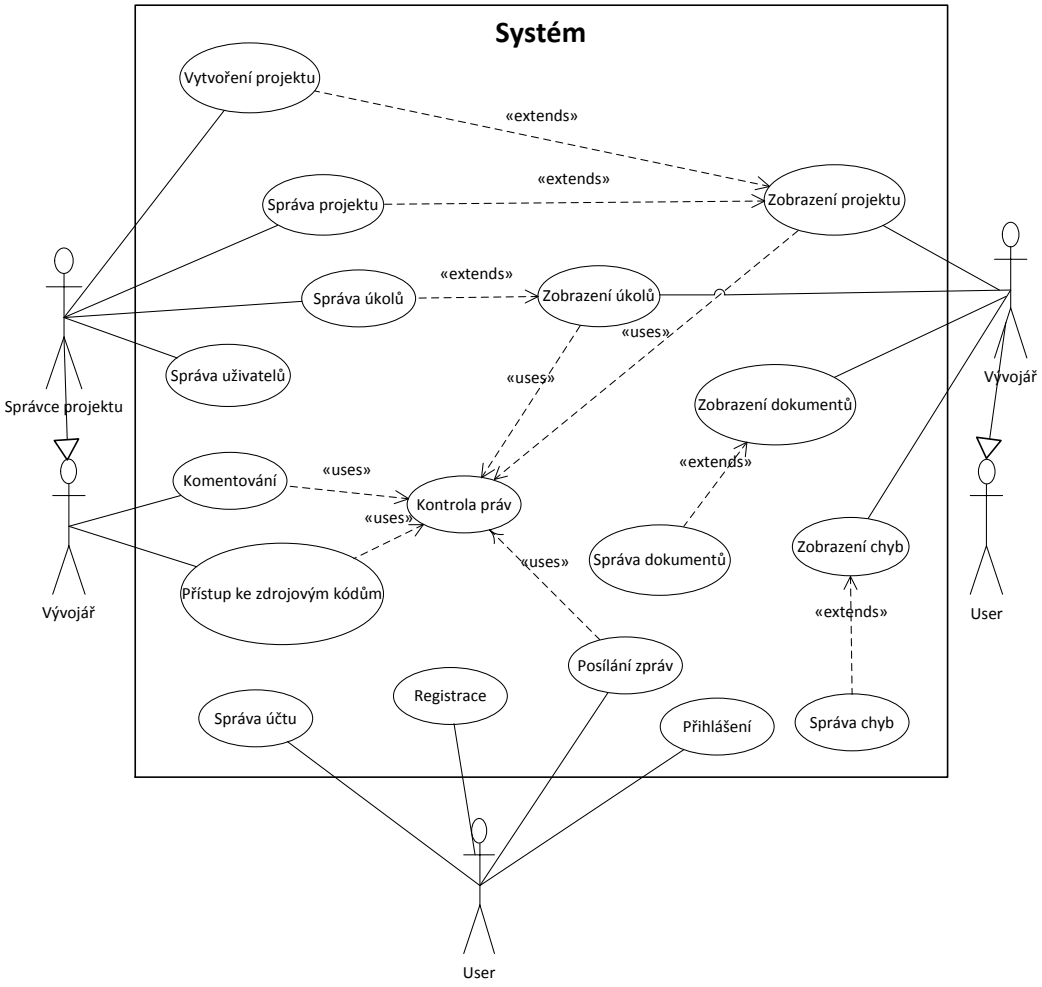
Kontexotvý diagram na obrázku 4 znázorňuje abstraktní pohled na systém, jeho vstupy a výstupy k extérním zdrojům. K aplikaci přistupují 3 typy uživatelů.



Obrázek 4: Kontextový diagram

4.4.3 Use Case diagram

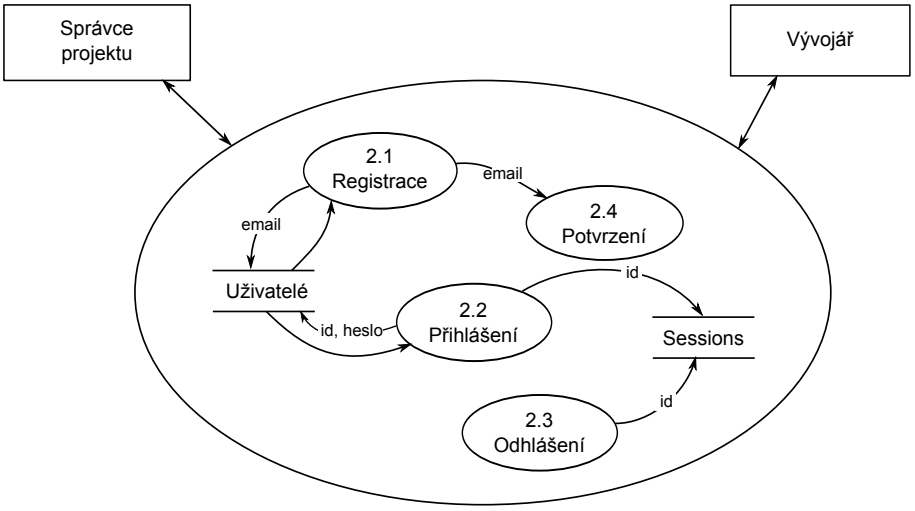
Diagram případů užití (Use Case diagram) zobrazuje chování systému z pohledu uživatele. Díky těmto diagramům dokážeme určit funkčnost jednotlivých částí aplikace, definovat omezení přístupu jednotlivým uživatelům, v jaké formě bude interakce aktérů probíhat a především odhalit hranice výsledného systému. Use case diagram se skládá z „aktérů“ a „případů užití“. Aktéři, jejichž symbolem je panáček, charakterizují konkrétní typy uživatelů a extérních zdrojů, které budou komunikovat s aplikací. Případy užití se označují oválem a definují posloupnost akcí ve vztahu s aktérem. Měly by vyjadřovat, co od vztahu očekávají aktéři, ne jestli systém jejich požadavkům vyhoví. Use case diagram je znázorněn na obrázku 5.



Obrázek 5: Use case diagram

4.4.4 Diagram datových toků

Diagram datových toků (DFD) slouží k zachycení vztahů mezi jednotlivými funkcemi, aktéry a datovými prvky. Diagram se skládá z procesů, toků, úložišť a aktérů. Proces představuje část systému - funkci, která mění vstupy na výstupy, symbolem je ovál. Tok znázorňuje přesun informací z jedné části do jiné, znázorňuje se šipkou. Uložiště vyjadřuje uložená data k pozdějšímu použití, v diagramu se značí dvěmi vodorovnými čárami. Poslední částí je aktér, ten představuje externí entitu komunikující se systémem a označuje se obdelníkem. Diagram datových toků hierarchicky rozčleňuje jednotlivé procesy až do fáze, kdy jsou dále nedělitelné, vzniká tedy určitý počet úrovní. Na obrázku 6 je znázorněn digram druhé úrovně, zbývající diagramy naleznete v příloze.



Obrázek 6: DFD - 2.úroveň

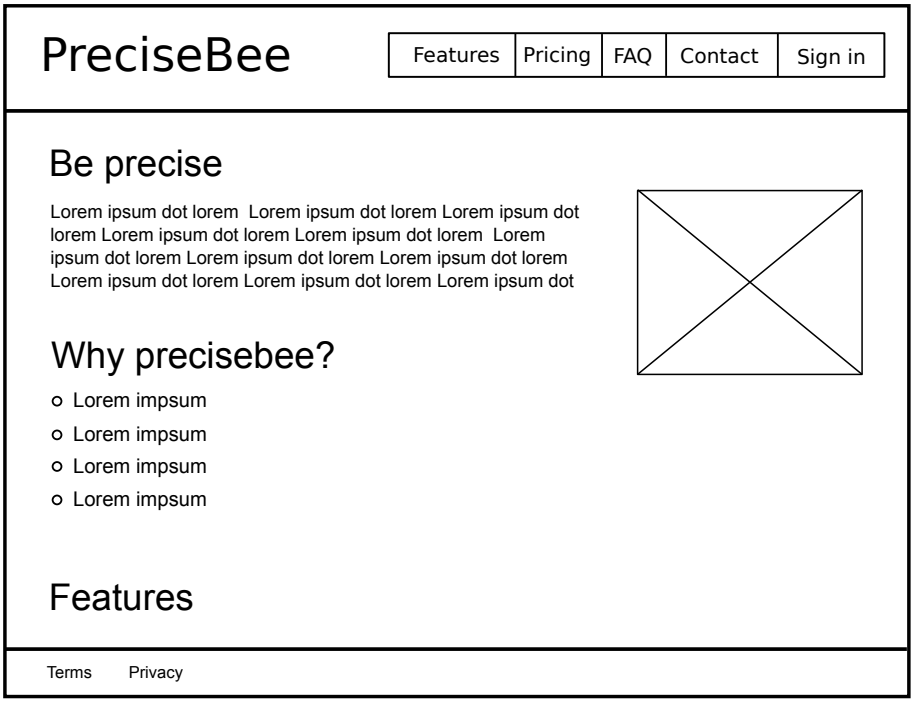
4.5 Volba názvu aplikace

Jako název aplikace jsem zvolil „PreciseBee“. Jde se o složení dvou anglických slov „precise“ a „bee“, v překladu „precizní“ a „včela“. Název tím má odkazovat na dokonalou organizaci při práci včel v úlech, tedy ať je stejným způsobem zorganizován i projekt. Navíc tím vznikla i zajímavá slovní hříčka v opačném pořadí slov, a sice slogan „be precise“. V češtině „buďte precizní“.

4.6 Návrh GUI aplikace

Grafické prostředí webu bude rozděleno na dvě nezávislé části:

Informační stránky Tato část se zobrazí uživateli po prvním přístupu na stránky. Návrh wireframu je vidět na obrázku 7. Uvodní stránka musí být přehledná a jasně popisovat,



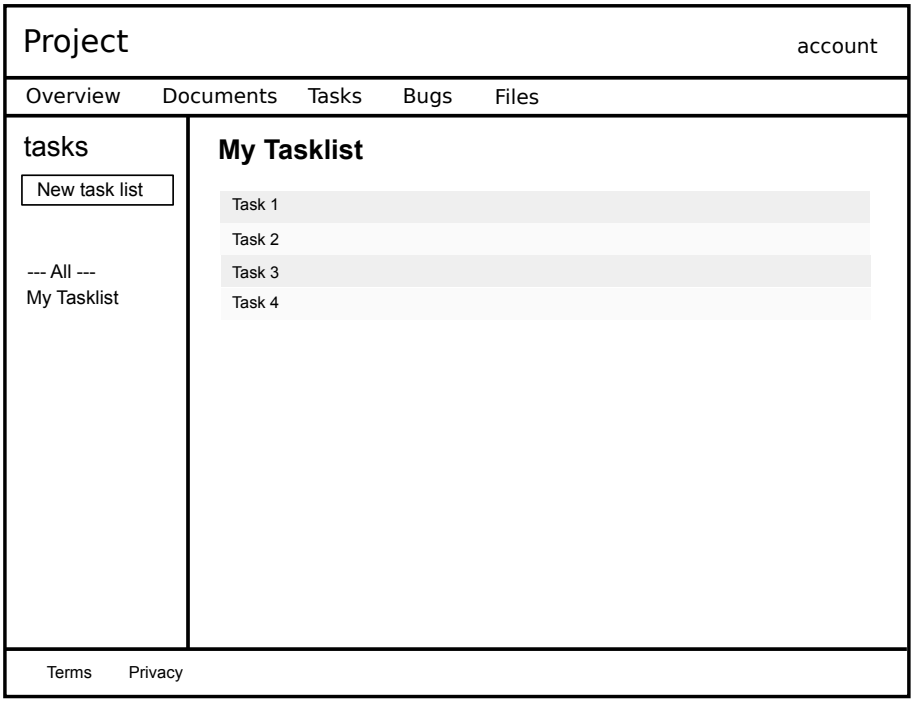
Obrázek 7: Grafický návrh hlavní stránky

o čem projekt je. Dále musí být dobře vidět odkazy pro registraci a přihlášení. Neregistrovanému návštěvníkovi stránek je vhodné také nabídnout seznam nejdůležitějších vlastností aplikace a způsoby plateb za užívání tak, aby se mohl dobře rozhodnout, nejlépe v náš prospěch.

Aplikace Druhou částí je již samotná aplikace. Od informačních stránek se podstatně liší, není již třeba uživatele zatěžovat informacemi nesouvisejícími přímo s jeho prací. Wirefram je znázorněn na obrázku 8. V horní části - hlavičce bude hlavní navigační menu a název aktuálního projektu, popřípadě jméno přihlášeného uživatele, pokud zrovna spravuje svůj účet. Pod hlavičkou je obsah rozdělen na dvě části. Vlevo se nachází druhé menu, jehož akce se mění podle právě zobrazované sekce (Dokumenty, úkoly, soubory...). V pravé části se potom nachází samotný obsah. V dolní části je nakonec patička, kde jsou umístěny další odkazy vztahující se k celému webu a aplikaci.

4.7 Způsoby monetizace

Není od věci již při návrhu projektu myslet také na způsob monetizace, tím více, pokud chceme, aby nás případně živil. Pro webové služby se používají především následující způsoby:



Obrázek 8: Grafický návrh stránky aplikace

Placený přístup

Jeden z nejvýnosnějších způsobů. Uživatel si musí funkce, popřípadě celý přístup zaplatit. Ideální je nabídnout mu omezenou funkčnost, aby si mohl aplikaci vyzkoušet. Pokud se mu služba zalíbí, nemusí pro něj být až takový problém přejít na jeden z placených modelů.

Reklamy

Málo ziskové řešení. Aplikace v tomto případě vydělává na počtu zobrazení reklamy nebo počtu kliknutí. Celkový zisk je poté velmi malý v porovnání s přímým placením za přístup. Může ale fungovat v kombinaci, kdy se reklamy budou zobrazovat pouze u uživatelů užívajících aplikaci zkušebně zdarma. Rozhodně ale nesmí zobrazování působit moc agresivně.

Affilate prodej

Affilate prodej znamená zprostředkování prodeje jiného produktu. Zpravidla jde o odkaz umístěný na webu. Když na něj uživatel klikne, je přesměrován na stránky prodejce a pokud tato návštěva povede k zakoupení výrobku/služby, získává zprostředkovatel provizi. Pro tuto bakalářskou práci affilate prodej uplatnění zřejmě nemá.

5 Návrh a implementace aplikace PreciseBee

Pro vývoj systému jsou používány opensource programy a komponenty. Zastávám názor, že nemá smysl znovu objevovat kolo. Pokud jsem tedy našel volně dostupné řešení problému a licence to umožňovala, použil jsem jej při vývoji. To se týká hlavně javascriptových knihoven a pluginů.

5.1 Použité technologie

K vývoji samotné aplikace jsem zvolil hlavně technologie, které znám a mám s nimi zkušenosti. Diskutabilní může být použití nerelační databáze MongoDB. K tomu mě vedla především zvědavost, zda samotný proces realizace bude náročnější nebo lehčí v porovnání s použitím relační databáze.

5.1.1 Python

Python je multiplatformní, dynamický, objektově orientovaný programovací jazyk určený pro efektivní vývoj aplikací. Vyznačuje se především striktní, jednoduchou a čitelnou syntaxí, díky které je často doporučován začínajícím programátorům. Python je hybridní jazyk, umožňuje při psaní používat kromě objektově orientovaného paradigmatu také paradigma procedurální nebo částečně funkcionální. Vývojář si tedy může zvolit návrh psaní programu podle jeho sympatií, nebo dle způsobu, který se lépe hodí k řešené úloze. Výsledný kód bývá zpravidla kratší a čitelnější než v jiných jazycích, což mimo jiné zvyšuje produktivitu práce [8].

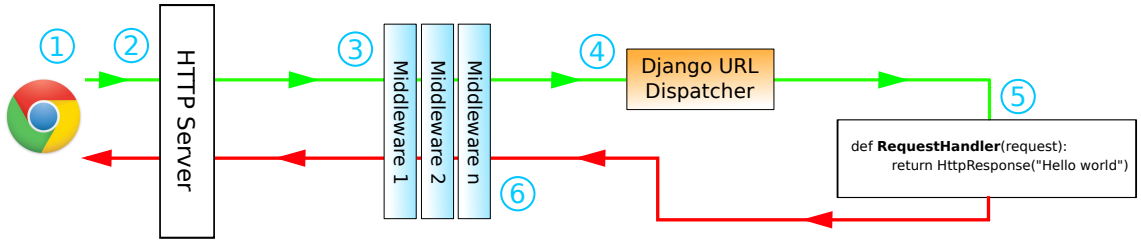
5.1.2 Django

Jako webový framework jsem zvolil Django, jeden z nejznámějších a nejpoužívanějších frameworků pro Python. Django je vyspělý open source projekt umožňující rychlý vývoj webových aplikací. Mezi některé zajímavé vlastnosti [9]:

- je postaven na modelu MVC
- obsahuje vlastní ORM
- automaticky vytváří administrátorské rozhraní
- nabízí efektivní a rozšiřitelný jazyk pro tvorbu šablon
- plně podporuje návrh vícejazyčných aplikací
- obsahuje vlastní debug server pro snažší vývoj

Proces zpracování HTTP požadavku je znázorněn na obrázku 9.

1. Klientův internetový prohlížeč pošle klasický HTTP požadavek na webový server.



Obrázek 9: Proces zpracování HTTP požadavku

2. Server předtím načte do paměti několik kopií instance webové aplikace připravených ke zpracování požadavku.
3. Django vytvoří objekt Request vyjadřující požadavek klienta a přefiltruje jej přes nastavené „middlewares“. Middleware je funkce vytvořená v pythonu, která dokáže požadavek nebo odpověď upravit pro další zpracování.
4. Django ověří podle regulérních výrazů definovaných v souboru urls.py požadovanou adresu a zavolá přiřazenou funkci.
5. Poté již následuje zpracování požadavku controllerem. Funkce dostává jako parametr upravený Request objekt, provádí potřebné změny a vrací objekt Response.
6. Objekt Response naposledy prochází přes middlewares, kde může být ještě upraven. Poté již klientovi přichází finální odpověď serveru.

5.1.3 MongoDB

Jako úložný sytém jsem zvolil databázi MongoDB. Jedná se o dokumentově orientovanou, lehce škálovatelnou a vysoce výkonnou open source databázi spadající do kategorie tzv. NoSQL databází. Způsob úschovy dat a dotazování se tedy poměrně liší od databází relačních, dotazy jsou psány v jazyce Javascript. MongoDB uchovává údaje v BSON (binárně kódovaných JSON objektech), nabízí automatické horizontální škálování, snadnou replikaci, rychlou úpravu dat, journaling. Bohužel daní za rychlost je absence transakcí a složitější agregační funkce.

Nejlepší užití je v případech, že potřebujete vykonávat dynamické dotazy, vyžadujete dobrý výkon pro velké množství informací a pokud například nechcete, aby vás při vývoji zdržovalo předdefinování sloupců tabulek u relačních databází. Databázový systém je napsán v C++ a užívá paměťově mapované soubory pro ukládání dat. Prázdná databáze má necelých 200 MB a zaplněna bude mít velikost pravděpodobně vyšší než srovnatelná databáze v relačních systémech. To je dáno právě možností libovolné změny schématu databáze za běhu. Proto MongoDB vymezí pro kolekci více místa, než je aktuálně potřeba. Vyhne se tím časté realokaci při úpravě schématu databáze, ale i při úpravě dat v existující dokumentu [10].

Protože framework Django nativně nepodporuje NoSQL databáze, zvolil jsem jako alternativní ORM, pro mne již v několika projektech osvědčený, Mongoengine. Některé

SQL Dbs	MongoDB
Database	Database
Table	Collection
Row	BSON Document
Column	BSON Field
Index	Index
Join	Linking & Embedding
Primary Key	_id field
Group By	Aggregation
Partition	Shard

Tabulka 3: Srovnání SQL databází a MongoDB

rozdíly a porovnání jsou znázorněny v tabulce 3. Zajímavou vlastností je tzv. „Embedding“. Ten se dá představit jako vložení dokumentu do dokumentu. Hodnota atributu kolekce neodkazuje na referenci jiné kolekce (analogie relačních databází je spojování pomocí cizích klíčů), ale data jsou stromovou strukturou obsaženy přímo jako atribut kolekce. Pro lepší představivost je takový jednoduchý dokument znázorněn v kódu 1. Schéma popisuje jednoduchý článek s autorem, popiskem článku, textem a komentáři. Právě komentáře jsou pak definovány jako pole vložených dokumentů. Výbornou vlastností MongoDB je, že se nad těmito vloženými dokumenty dají vytvářet indexy a snadno je dotazovat. Každá kolekce má pak svůj ObjectID neboli primární klíč, složený z časové známky, identifikátoru serveru, procesu a inkrementující části.

```
{
    _id : ObjectId("4e77bb3b8a3e000000004f7a"),
    autor : "alex",
    popisek : "Revolucni metoda vyhledavani",
    text : "Vedci z americke univerzity zjistili ... ",
    comments : [
        { who : "Karel", when : Date("2011-09-19T04:00:10.112Z"),
            comment : "Neverim." },
        { who : "Jana", when : Date("2011-09-20T14:36:06.958Z"),
            comment : "To se mi nejak nezda" }
    ]
}
```

Výpis 1: Ukázka embedded dokumentu v MongoDB

5.1.4 Git

Git je distribuovaný systém pro správu a verzování zdrojových kódů. Git přistupuje k datům jako k sadě snímků vlastního malého systému souborů, v podstatě tak stav projektu při každé změně jakoby „vyfotí“. Tím se liší od ostatních verzovacích systémů, které data většinou uchovávájí jako seznam změn. Pro většinu operací potřebuje Git pouze lokální soubory, na server se nahrávají pouze změny. Díky tomu probíhají úkony téměř

okamžitě. Původně byl Git vytvořen Linusem Torvaldem pro správu vývoje linuxového jádra. Úložiště může být zveřejněno pomocí protokolů HTTP, FTP, rsync nebo Git realizovaného buď přes obyčejné sokety nebo přes ssh. V této aplikaci má uživatel přístup k repozitáři projektu pomocí šifrovaného SSH tunelu [7].

5.1.5 HTML5 / CSS / Javascript

Uživatelské prostředí jsem se rozhodl vytvořit jako klasické webové stránky ve specifikaci HTML5. HTML je hlavní značkovací jazyk pro tvorbu www stránek. Narozdíl od jiných RIA řešení, jako jsou např. Adobe Flex, Microsoft Sivlerlight, je při tvorbě webových aplikací zajištěno, že bude fungovat na většině moderních zařízení nezávisle na použité platformě.

5.1.6 CSS

CSS neboli Caascading Style Sheets (česky "Kaskádové styly") je jednoduchý jazyk popisující způsob zobrazení stránek napsaných v HTML, XHTML nebo XML. Hlavním smyslem je oddělení vzhledu stránek od jejich struktury a obsahu.

5.1.7 Javascript

Interaktivní prvky na stránkách, vykreslování a především asynchronní komunikace je řešena pomocí jazyka Javascript v kombinaci s frameworkem jQuery. Javascript je objektově orientovaný skriptovací jazyk vytvořený společnosti Netscape, nyní zpravidla používaný pro tvorbu webových stránek. Jeho syntaxe je podobná rodině jazyků C/C++/Java. Se zmíněnou Javou nemá ovšem kromě syntaxe a názvu nic společného.

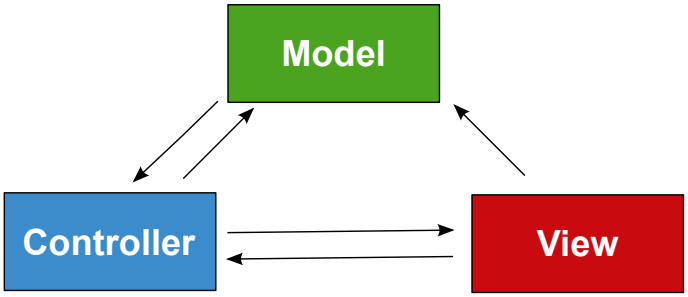
Zmíněné jQuery je lehký javascriptový framework usnadňující práci s DOM elementy stránky.

5.2 Vývojové prostředí

Celý vývoj aplikace byl realizován v prostředí Eclipse pro operační systém linux. Tento editor je vhodný především díky velkému množství dostupných pluginů. Dají se v něm pohodlně vytvářet HTML stránky a pomocí pluginu pyDev i editovat zdrojové soubory Pythonu. Pro úpravu grafiky jsem používal grafický editor Gimp a vektorový editor Inkscape.

5.3 Návrhový vzor MVC

Systém je vytvořen pomocí architektury MVC. Tato architektrura rozděluje uživatelské rozhraní, řídíci logiku a datový model aplikace do tří nezávislých komponent tak, že modifikace jedné z nich má minimální vliv na ostatní. „Model“ reprezentuje data a business logiku aplikace, „View“ zobrazuje uživatelské rozhraní a „Controller“ má na starost aplikační logiku a tok událostí [11].



Obrázek 10: Návrhový vzor MVC

Vývojáři frameworku Django tvrdí, že jejich produkt poskytuje architekturu MTV (model - template - view). Template zde nahrazuje vrstvu View a View naopak znamená Controller. Toto přejmenování má na svědomí fakt, že u webových aplikací se za uživatelské rozhraní považují html stránky, které se tvoří právě pomocí šablon. Jedná se ale o stejnou architekturu jako je MVC, jen jinak pojmenovanou.

5.4 Adresářová struktura

Adresářová struktura odpovídá vygenerovanému stromu frameworkem Django a základním bezpečnostním zásadám.

- **softman** - kořenový adresář hlavní aplikace automaticky vytvořeným frameworkem Django
  - **smapp** - adresář aplikace
    - *__init__.py*
    - *models.py* - soubor obsahující datové modely
    - *test.py* - soubor obsahující testy
    - *views.py* - soubor obsahující controllery
  - **utils** - adresář s pomocnými skripty
    - *__init__.py*
    - *auth.py* - skript pro logiku autorizace uživatelů
  - *__init__.py*
  - *manage.py* -
  - *settings.py* - skript s nastavením, obsahuje přihlašování k databázi, cesty k adresářům...
  - *urls.py* - skript popisující dostupné URL adresy aplikace, routování
- **static** - adresář statických souborů
  - **css** - adresář se soubory css stylů užívaných v šablonách

  - **img** - adresář s obrázky pro html stránky
  - **js** - adresář s javascriptovými soubory
- **templates** - adresář s šablonami html stránek (u MVC se jedná o vrstvu View)
- **files** - adresář sloužící k nahrávání souborů k projektům uživateli

Hlavní adresář a jeho stromovou strukturu automaticky vygeneruje Django, je nutno ji dodržet. Adresář „static“ je jediný přímo přístupný, protože neobsahuje citlivá data a je tedy zbytečné zatěžovat aplikaci servírováním jeho obsahu.

5.5 Ukázka modelu

Jedním z prvních úkonů při vývoji bylo vytvoření modelů podle analýzy. O ukládání do databáze jsem se nemusel starat, toto řeší ORM mongoengine.

```
class Project(Document):
    name = StringField(max_length=30)
    slug_name = StringField()
    created = DateTimeField(default=datetime.utcnow())

    managers = ListField(ReferenceField(User))
    developers = ListField(ReferenceField(User))

    terms = ListField(EmbeddedDocumentField(Term))
    term_counter = IntField(default = 1)

    def __unicode__(self):
        return self.name

    def add_term(self, term):
        term.term_id = self.term_counter
        self.terms.append(term)
        self.term_counter += 1
        self.save()

    def get_term(self, term_id):
        return self.__findTerm(term_id)
```

Výpis 2: Ukázka modelu User

Ve výpisu 2 je pro ukázku část datového modelu Project obsahující i metody pro přidávání termínů. Atributy a metody se definují pouze v této třídě, není potřeba upravovat žádné databázové schéma. Třída dědí z mongoengine třídy Document. Ta obsahuje další atributy jako _id dokumentu, či metody pro ukládání a odstranění.

5.6 Ukázka controlleru

Mezikrok při prezentaci datového modelu zajišťuje controller. V Djangu jsou to obyčejné metody s parametrem request umístěné v souboru views.py, které volá jádro Djanga po-

dle požadované URL. Controllery v této aplikaci jsou rozděleny do čtyř skupin podle prefixu:

- **main_*** - starají se o část převážně statických stránek informativní části webu
- **me_*** - obsluhuje volání spojené s uživatelským účtem v aplikaci
- **project_*** - obsluhuje volání části spojené s projektovou části v aplikaci
- **ajax_*** - obshluhuje ajaxovou komunikaci

Ve výpisu 3 je ukázán controller pro přihlášení uživatele. Podle typu požadavku buď zobrazí stránku pro přihlášení uživatele nebo provede proces přihlášení s následným přesměrováním.

```
@csrf_protect
def signin(request):
    """ Prihlaseni uzivatele """
    if request.method == 'GET':
        return render(request, 'signin.html', {})

    elif request.method == 'POST':
        email = request.REQUEST.get('email', None)
        passw = request.REQUEST.get('pass' ,None)
        user = authenticate(email=email, password=passw)

        if user:
            login(request, user)
            return redirect('/me/')
        else:
            return redirect('/account/signin')
```

Výpis 3: Ukázka controlleru pro přihlášení uživatele

5.7 Ajaxová volání

K použití asynchronní komunikace na webu může mít následující důvody:

- Poskytnutí uživateli funkce, které usnadní nebo urychlí práci (např. našeptávače).
- Omezit množství stahovaných dat při návštěvě webových stránek. Při asynchronním volání se odesíla jen požadavek a klient stahuje odpověď ve formátu JSON nebo XML, která je řádově menší než stáhnutí znovu celé stránky.
- Přiblížení chování webu desktopovým aplikacím. Služba pak asynchronně komunikuje se serverem s požadavkem na různá data a oprávnění, klientský prohlížeč je pak zpracuje a vykreslí uživateli.

Ajax využívám k triviálním akcím, u kterých nemá smysl znovu načítat a vykreslovat celou stránku. Řešeno je tak například získávání dat k editaci příspěvků, mazání, změny stavu úkolu (dokončeno / nedokončeno), našeptávání pro příjemce zprávy. Komunikaci

vyvolává klient pomocí frameworku jQuery. Ukázka kódu pro změnu stavu úkolu je ve výpisu 4. JQuery odešle na pevnou URL pro ajaxové funkce data „action“ a „task_id“ a čeká na odpověď serveru ve formátu JSON. Pokud je poté status roven "ok", změní se označení úkolu.

```
function markAsDone(object, tasklist_id, task_id) {
    \$.ajax({
        url: "/project/{{ project.id }}/ajax/",
        data: {
            "action" : "changeDoneTask",
            "task_id" : task_id
            },
        success: function(data) {
            if (data.status == 'ok') {
                \$(object).closest('dd').toggleClass('done');
            }
            else if (data.status == 'error') {
                alert("Na serveru se vyskytla chyba")
            }
        }
    });
}
```

Výpis 4: Odeslání asynchronního požadavku pomocí jQuery

Takových funkcí volání je poměrně dost a nemá smysl vytvářet pro každou vlastní controller. Proto je definován pouze jeden univerzální (znázorněna ve výpisu 5), který zprostředkovává požadovanou odpověď. Nejprve zjistí, zda požadovaná akce vůbec existuje a jestli není privátní. Poté vytvoří instanci třídy AjaxManager a zavolá funkci této instance jako je název požadované akce.

```
@login_required
def ajax_project(request, project_id):
    action = request.REQUEST.get('action', None)

    #overeni, zda pozadovana akce neni privatni
    if action and action[0] != '_':
        am = AjaxManager(project_id, request)
        func = getattr(am, action)
        return func()
    else:
        return HttpResponse(simplejson.dumps({'status' : 'error'}), mimetype='application/json')
```

Výpis 5: Controller pro obsluhu ajaxu

AjaxManager poté zpracuje data požadavku a vrací zpátky odpověď ve formátu JSON. Část kódu je znázorněna ve výpise 6.

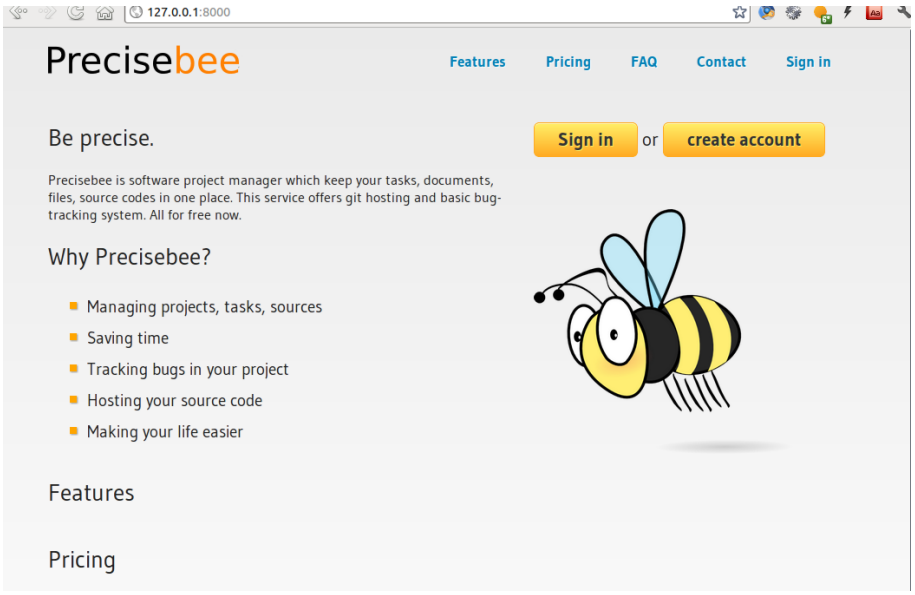
```
class AjaxManager():
    def __init__(self, project_id, request):
        self.project_id = project_id
        self.request = request
        self.req = request.REQUEST
```

```
def _response_json(self, status = 'ok', data = None):
    response = {'status' : status}
    response.update(data) if data else None
    return HttpResponse(simplejson.dumps(response), mimetype='application/json')

def changeDoneTask(self):
    task_id = self.req.get('task_id', None)
    task = Task.objects(id=task_id).first()

    if task:
        task.change_done()
        return self._response_json('ok')
    else:
        return self._response_json('error')
```

Výpis 6: Třída pro obsluhu ajaxu

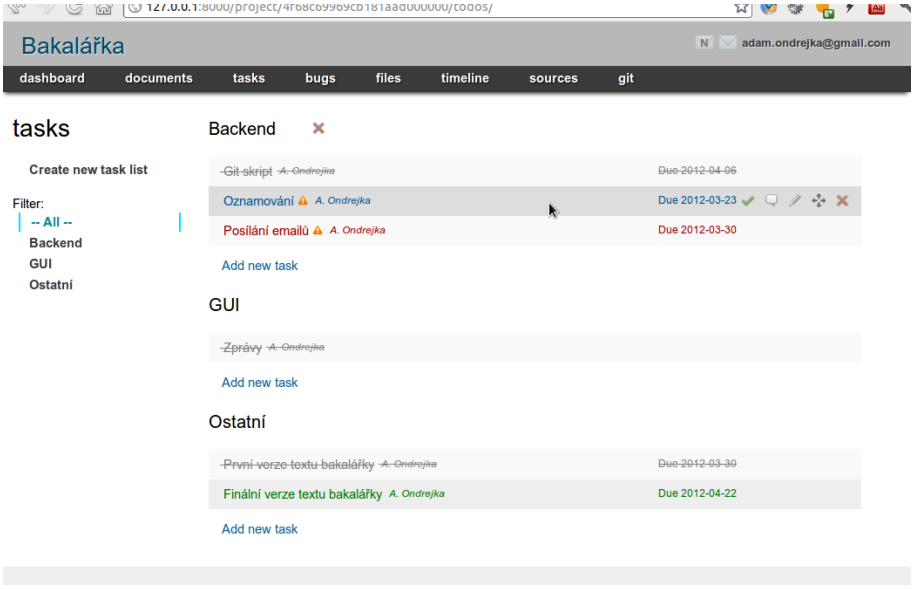
5.8 Implementace grafického návrhu

Nejdříve jsem navrhnul design stránky v grafickém editoru Gimp a poté tento návrh implementoval pomocí technologií HTML5 a CSS3. V prohlížečích, které tyto nejnovější standardy nepodporují, se některé funkce mohou zobrazovat trochu jinak. Jedná se ale jen o rozdíly v grafice.



Obrázek 11: GUI Informativní stránky

Informativní stránka je na obrázku 11, vzhled samotné aplikace můžeme vidět na obrázku 12. Nutno podotknout, že výtvarnictví není má silná stránka. Skutečný grafik by si se vzhledem pohrál určitě více a výsledek by vypadal mnohonásobně lépe.



Obrázek 12: GUI stránky s úkoly

5.9 Produkční nasazení

Webovou aplikaci je potřeba hostovat na vlastním serveru. Pro první nenáročné nasazení poslouží VPS s operačním systémem Ubuntu Server 10.04. Provoz vyžaduje minimálně tyto aplikace:

Python 2.7, Django 1.3, mongoengine 0.6

Jazyk Python v kompatibilní verzi s frameworkem Django 1.3. Dále ORM mongoengine ve verzi 0.6, která je plně kompatibilní s používánou verzí databáze.

nginx

Nginx je populární, „lehký“ a výkonný webový server. V kombinaci s djangem se používá pro servírování statických souborů, pro které webový server Apche není příliš vhodný, a poslouží jako reverzní proxy. Bude tedy naslouchat na portu 80 a veškeré požadavky, mimo stahování statických souborů, přesměrovávat na webový server Apache.

Apache web server 2.0

Webové aplikace psané v Pythonu se dají hostovat v několika režimech (CGI, FastCGI...). Mezi dnešní nejpoužívanější patří komunikační protokol WSGI, vytvořený speciálně pro potřeby Pythonu. Mezi jeho přednosti patří především rychlost a jednoduchost. Server Apache v tomto případě naslouchá na portu 8080, požadavky přeposílá server nginx.

MongoDB 2.0

Databáze poběží v daemon (skrýtém) režimu. Verze 2.0 již obsahuje některé užitečné agregační funkce.

Memcached

Neméně důležitý je také cachovací systém pro zrychlení běhu aplikace. Memcached je jednoduché key-value úložiště běžící odděleně od hlavní aplikace. Data se ukládají pouze do dostupné operační paměti, což zaručuje vysokou rychlost přístupu k datům. Při pádu následuje ztráta dat, z tohoto důvodu není vhodný jako alternativa databázového systému.

5.10 Škálovatelnost aplikace

Návštěvnost webu je přímo úměrná naší spokojenosti, potencionálního zisku, ale bohužel také starostí. Při stovkách požadavků během jedné vteřiny pravděpodobně nebude řešení na jednom virtuálním serveru nejrychlejší. Zvyšuje se tím také šance na pád jednotlivých částí systému, což zapříčíní pád celé aplikace. Ano, dá se na serveru omezit počet přístupů, z toho ale uživatelé určitě spokojeni nebudou. K vyřešení této situace se dá v budoucnu přistupovat dvěma způsoby:

5.10.1 Cloud hosting

Velmi oblíbené slovo poslední doby „cloud“ se nevyhlo ani hostingu. Cloud hosting by se dal vyjádřit jako virtuální hosting založený na obrovské infrastruktuře, kde jsou výpadky téměř zcela eliminovány. Kouzlo tkví v tom, že se poté platí pouze za spotřebovaný výpočetní čas, obsazené místo, spotřebovanou pamět. Záleží na nastavení služby. Tento způsob hostování nabízí například Amazon, Google a jeho Appengine či Microsoft s Azure. Pro chod této bakalářské práce bychom museli hledat hosting zvaný jako IAAS, nebo-li Infrastructure as a Service, která virtualizuje celou infrastrukturu.

Výhody

- Nulové počáteční náklady.
- Snadnější škálovatelnost aplikace.
- Malá šance nepřístupnosti webu.
- Platba jen za skutečně využité služby.

Nevýhody

- Data nejsou pod vlastní kontrolou.
- Přechod na vlastní infrastrukturu může být složitější (záleží na konkrétním hostingu).
- Po určité době je provoz dražší než provoz vlastní infrastruktury.

5.10.2 Vlastní infrastruktura

Druhým řešením je vybudování vlastní infrastruktury. V takovém případě je nutno počítat s investicí do serverů a paušálním poplatkem za umístění v datacentru (pokud nemáme vlastní) a náklady spojené s provozem a údržbou. Je vhodné mít infrastrukturu předimenzovanou, aby bez problému ustála nárazový nápor návštěvníků.

Výhody

- Data jsou na vlastní infrastruktuře pod lepší kontrolou.
- Po určité době vychází provoz vlastní infrastruktury v porovnání s cloud hostingem levněji.

Nevýhody

- Větší prvotní náklady.
- Náročnější na údržbu.
- Při nedostatečném výpočetním výkonu hrozí výpadky při náporové návštěvnosti.

5.11 Git a práva uživatelů

Je důležité, aby zdrojový kód projektu byl přístupný jen určeným uživatelům a přenos byl dostatečně zabezpečen. Pro tento případ je vhodné použít spojení pomocí SSH tunelu, což zajistí bezpečný přenos dat bez možnosti „odposlouchávání“ uživatelů. Dále je nutno delegovat práva uživatelů k jednotlivým projektům (repozitářům). Tato vlastnost bohužel původnímu git systému chybí. Proto vznikly dva projekty „gitosis“ a „gitolite“, které dokaží omezovat uživatelům, popřípadě skupinám, práva ke čtení a zápisu k repozitářům, ale řešení správy a konfigurace repozitářů není vhodné pro tuto bakalářskou práci.

Vytvořil jsem tedy na serveru uživatele git. Ten slouží k obsluze zdrojových kódů, v jeho domovském adresáři se uchovávají všechny repozitáře. Uživatel aplikace si musí vytvořit na svém klientském počítači soukromý a veřejný SSH klíč. Ten poté nastaví pomocí webového rozhraní, aplikace tento kód uloží do databáze a přidá do seznamu autorizovaných klíčů. Tento seznam obsahuje také u každého klíče automaticky vygenerovaný příkaz, který spustí skript po úspěšném vytvoření spojení mezi klientem a serverem. Skript (vytvořený v Pythonu) zjistí, jestli klient spouští povolený příkaz a podle předaného parametru najde v databázi daného uživatele a ověří, zda má příslušné práva k repozitáři. Pokud ano, spustí původní příkaz, jinak zakáže spojení.

5.12 Implementace Ganttova diagramu

Při hledání užitečných funkcí projektového řízení jsem se rozhodl pro implementaci Ganttova diagramu (viz. kapitola 3). Protože jsem nenašel žádné vhodné volně dostupné řešení, volba padla na vytvoření řešení vlastního. To jsem nakonec realizoval jako jQuery

plugin, který vykresluje diagram pomocí standardních HTML prvků. Základem jsou dva hlavní panely, levý s fixní pozicí pro zobrazování názvů úkolů a pravý posuvný panel pro vizualizaci časových údajů úkolů. Vstupem je datová strutruta skládající se z pole úkolů (tasks) a pole termínů - milníků (terms).

5.13 Testování ukázkového modelu

Mějme, jak již jsem uvedl v kapitole 2, malou firmu zabývající se vývojem webových aplikací. Tým se skládá ze 4 osob, každý má v systému vytvořen testovací účet:

- *Obchodník* - přihl. email: obchodnik@precisebee.com , heslo: obchodnik
- *Programátor* - přihl. email: programator@precisebee.com , heslo: programator
- *Grafik* - přihl. email: grafik@precisebee.com , heslo: grafik
- *Koder* - přihl. email: koder@precisebee.com , heslo: koder

Ukázková situace je následující. Obchodník po konzultaci s klientem na novou firemní webovou aplikaci specifikuje požadavky. Tento dokument nahraje do systému precisebee a pošle programátorovi zprávu o tom, ať na základě tohoto dokumentu provede analýzu. Mezitím obchodník, tedy i projektový manažer, zadá do systému milníky vztahující se k projektu a odhadne dobu trvání jednotlivých fází procesu (např. analýza, návrh, implementace, testování). Protože je projektový manažer zároveň šikovný webdesigner, navrhne základní strukturu stránek - wireframy, soubory nahraje do precisebee, vytvoří nový úkol a pošle grafikovi zprávu, ať na základě těchto návrhů zpracuje grafickou podobu webu.

Mezitím je programátor s analýzou hotov a projektový manažer mu přidělí další dílčí úkoly, na kterých může postupně pracovat. Programátor ovšem nerozumí jednomu ze zadaných úkolů, má nyní dvě možnosti: Buď pošle zprávu manažerovi s žádostí o objasnění, nebo vytvoří komentář přímo k úkolu. Rozhodne se pro druhou možnost, informace bude mít na jednom místě, což při případném zpětném hledání ušetří dosti práce.

Grafik při realizaci návrhu přijde na skvělý nápad, ale není si jim stroprocentně jist a potřeboval by ho prodiskutovat. Vytvoří proto nový dokument s popisem jeho myšlenky a umožní ostatním jej komentovat. Tím vznikne podnětná diskuze k nápadu, která ho nakonec přesvědčí o jeho správnosti, proto ho do návrhu zahrne. Když je s grafikou hotov, pošle zprávu kodérovi, ať vše převede do html a projektový manažer sleduje, zda vše pokračuje tak, jak má a zda-li se stíhá práce v termínech. Programátor pracuje na svém kódu a změny nahrává do git repozitáře v Precisebee.

Po dokončení implementace nastává fáze testování. Do něj se v našem případě zapojí grafik a kodér. Grafik po objevení chyby v implementaci jeho návrhu zadá chybu do systému a projektový manažer opravu přidělí kodérovi. Ten po opravě označí chybu za vyřešenou. Po vyřešení všech chyb a odsouhlašení klientem se projekt uzavírá jako dokončený.

6 Závěr

Cílem práce bylo vytvořit systém pro podporu týmového vývoje softwarových projektů. Výsledný systém měl mít podobu webové služby s intuitivním rozhraním.

V úvodu práce jsem stanovil cíle a obecně popsal správu a řízení projektu. Porovnal jsem a analyzoval konkurenční služby a při realizaci vlastního řešení se jimi částečně také inspiroval. Dále jsem zanalyzoval funkční a nefukční požadavky na webovou aplikaci a navhrnul uživatelské rozhraní.

Následně jsem na těchto základech navrhl a vytvořil základní aplikaci pro správu a řízení projektů. Vytvořil jsem správu úkolů s možností nastavování priority a přiřazování pracovníků, jednoduchý systém pro nahlašování chyb, vytváření dokumentů a jednoduché datové úložiště. Dále jsem realizoval funkce pro výměnu zpráv mezi členy týmů a komentování příspěvků. Pro přehlednost úkolů a jejich termínů jsem udělal vlastní plugin pro Ganttův diagram a realizoval zabezpečené hostování zdrojových kódů.

Pro obsluhu webových požadavků jsem zvolil webový framework Django, který velmi usnadňuje vývoj aplikací. Programovou část jsem vytvořil v dynamickém jazyce Python a užitelské rozhraní vytvořil pomocí HTML5, CSS3 a Javascriptu. Jako databázový systém jsem zvolil nerelační MongoDB, který se během realizace projektu neukázal jako příliš vhodný. To zejména kvůli složitosti vytváření reportů a spojování jednotlivých entit. Nakonec jsem výsledný software ověřil na modelové ukázce, kde se ukázal jako vyhovující.

Aktuální verze systému ještě nemůže plně konkurovat dostupným službám na trhu, lze ji ale bezproblémově používat. Stávající systém je vhodný pro veřejné beta testování. Aplikaci chci v začátku používat hlavně pro vlastní účely a dalším uživatelům ji nabízet, než se nevyladí, zdarma. Do budoucna plánuji rozšířit funkčnost aplikace, především vylepšit a sjednotit vzhled a zdokonalit použitelnost. Pro zlepšení kolaborativního přístupu bych rád zahrnul také verzování dokumentů a souborů a vytvořil vektorový editor pro tvorbu nákresů a diagramů. V případě kladných ohlasů od uživatelů pravděpodobně používání aplikace zpoplatním.

7 Reference

[1] Project Management Institute, *A guide to the project management body of knowledge (PMBOK guide)*. Newtown Square, Pa: Project Management Institute, 2000, 216 s. ISBN 18-804-1023-0.

[2] STANTON, Pam. *The Project Whisperer: Understanding The Human Part of The Gantt Chart* Heart, Brains & Courage LLC, 2011. ISBN: 978-0-98316-530-9

[3] *Basecamp.com* [online]. © 1999-2012 [cit. 2012-03-15]. Dostupné z: http://basecamp.com/

[4] *Projectial - project management software* [online]. © 2011 [cit. 2012-03-15]. Dostupné z: http://www.projectial.com/

[5] *Zoho projects - Project Management & Planning Software* [online]. © 2012 [cit. 2012-03-15]. Dostupné z: http://www.zoho.com/projects/

[6] *GitHub - Social Coding* [online]. © 2012 [cit. 2012-03-14]. Dostupné z: https://github.com/

[7] CHACON, Scott. *Pro Git*. New York: Apress, c2009, 265 s. Developer's library. ISBN 978-1-4302-1833-3.

[8] PILGRIM, Mark. *Dive into Python 3*. New York: Apress, c2009, 360 s. Expert's voice in open source. ISBN 14-302-2415-0.

[9] FORCIER, Jeff, Paul BISSEX a Wesley CHUN *Python Web Development with Django*. Upper Saddle River, NJ: Addison-Wesley, c2009, 377 s. Developer's library. ISBN 978-013-2356-138.

[10] MEMBREY, Peter, Eelco PLUGGE a Tim HAWKINS. *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. New Edition. New York, NY: Apress, 2010. ISBN 978-143-0230-519.

[11] DEACON, John. *Model-View-Controller Architecture*. [online]. 1995, 2009-05-01 [cit. 2012-04-02]. Dostupné z: http://www.jdl.co.uk/briefings/index.html#mvc

8 Příloha

1. **Všechny diagramy datových toků**, které jsou v elektronické podobě uloženy na CD.

2. **Disk CD** obsahující elektronickou verzi textu bakalářské práce, zdrojové kódy.